DNS: Domain Name System

People: many identifiers:
- SSN, name, Passport #

Internet hosts, routers:
- IP address (32 bit) - used for addressing datagrams
- "name", e.g., gaia.cs.umass.edu - used by humans

Q: map between IP addresses and name ?

Domain Name System:
- *distributed database* implemented in hierarchy of many *name servers*
- *application-layer protocol* host, routers, name servers to communicate to *resolve* names (address/name translation)
  - note: core Internet function implemented as application-layer protocol
  - complexity at network's "edge"

DNS name servers

Why not centralize DNS?
- single point of failure
- traffic volume
- distant centralized database
- maintenance

doesn't *scale!*

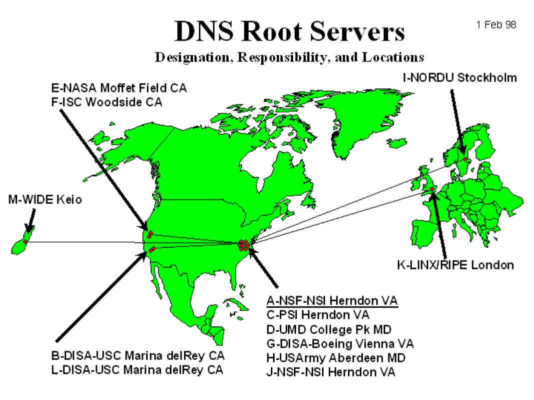
- no server has all name-to-IP address mappings

local name servers:
- each ISP, company has *local (default) name server*
- host DNS query first goes to local name server

authoritative name server:
- for a host: stores that host's IP address, name
- can perform name/address translation for that host's name

DNS: Root name servers

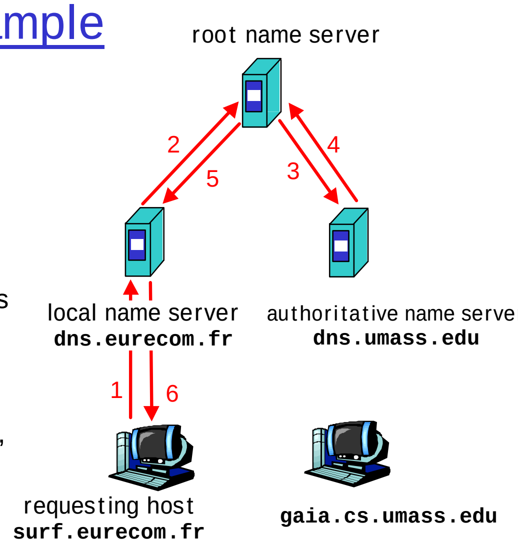
- contacted by local name server that can not resolve name
- root name server:
  - contacts authoritative name server if name mapping not known
  - gets mapping
  - returns mapping to local name server
- ~ dozen root name servers worldwide



Simple DNS example

host **surf.eurecom.fr** wants IP address of **gaia.cs.umass.edu**

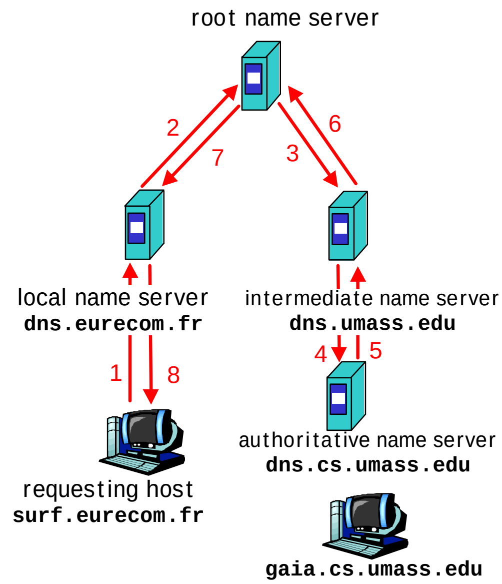
1. Contacts its local DNS server, **dns.eurecom.fr**
2. **dns.eurecom.fr** contacts root name server, if necessary
3. root name server contacts authoritative name server, **dns.umass.edu,** if necessary



DNS example

Root name server:

- may not know authoritative name server
- may know *intermediate name server:* who to contact to find authoritative name server



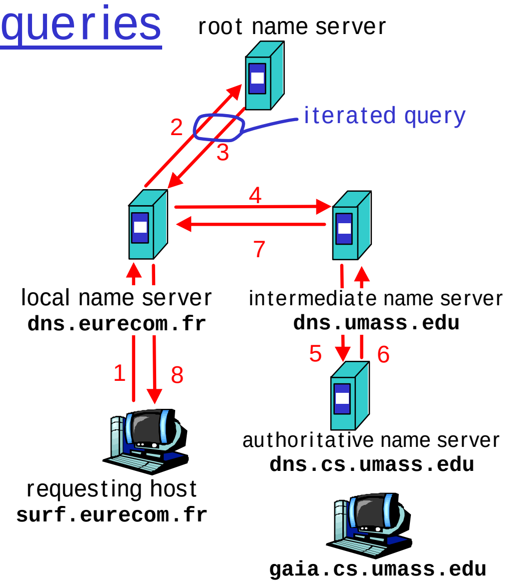
DNS: iterated queries

recursive query:

- puts burden of name resolution on contacted name server
- heavy load?

iterated query:

- contacted server replies with name of server to contact
- “I don't know this name, but ask this server”



DNS: caching and updating records

- once (any) name server learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time
- update/notify mechanisms under design by IETF
  - RFC 2136
  - http://www.ietf.org/html.charters/dnsind-charter.html

DNS records

DNS: distributed db storing resource records (RR)

RR format: `(name, value, type, ttl)`

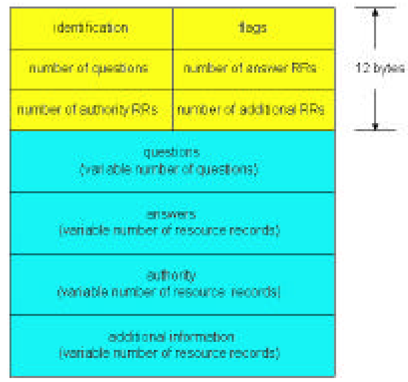
- Type=A
  - **name** is hostname
  - **value** is IP address
- Type=NS
  - **name** is domain (e.g. foo.com)
  - **value** is IP address of authoritative name server for this domain
- Type=CNAME
  - **name** is an alias name for some "canonical" (the real) name
  - **value** is canonical name
- Type=MX
  - **value** is hostname of mailserver associated with **name**

DNS protocol, messages

DNS protocol : *query* and *repy* messages, both with same *message format*

msg header

- identification: 16 bit # for query, reply to query uses same #
- flags:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative



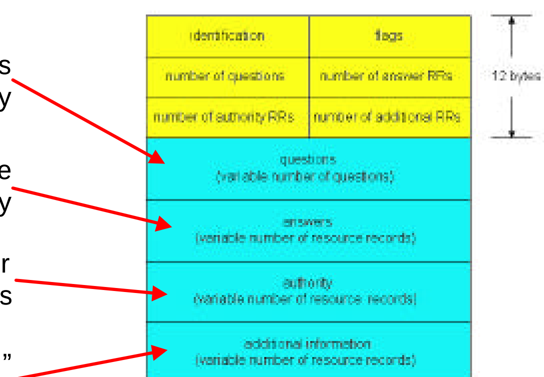
DNS protocol, messages

Name, type fields for a query

RRs in response to query

records for authoritative servers

additional "helpful" info that may be used



DNS Query Example:

```
Bayou.UH.EDU> nslookup
Default Server:  Masala.CC.UH.EDU
Address:  129.7.1.1

> www.yahoo.com
Server:  Masala.CC.UH.EDU
Address:  129.7.1.1

Non-authoritative answer:
Name:    www.yahoo.akadns.net
Addresses:  216.32.74.53, 216.32.74.55, 216.32.74.50, 216.32.74.51
          216.32.74.52
Aliases:  www.yahoo.com

> set querytype=ANY
> www.yahoo.com
Server:  Masala.CC.UH.EDU
Address:  129.7.1.1

Non-authoritative answer:
www.yahoo.com   canonical name = www.yahoo.akadns.net

Authoritative answers can be found from:
YAHOO.com       nameserver = ns1.YAHOO.com
YAHOO.com       nameserver = ns3.europe.YAHOO.com
YAHOO.com       nameserver = ns5.dcx.YAHOO.com
ns1.YAHOO.com   internet address = 204.71.200.33
ns3.europe.YAHOO.com    internet address = 194.237.108.51
ns5.dcx.YAHOO.com       internet address = 216.32.74.10
```


Socket programming

Goal: learn how to build client/server application that communicate using sockets

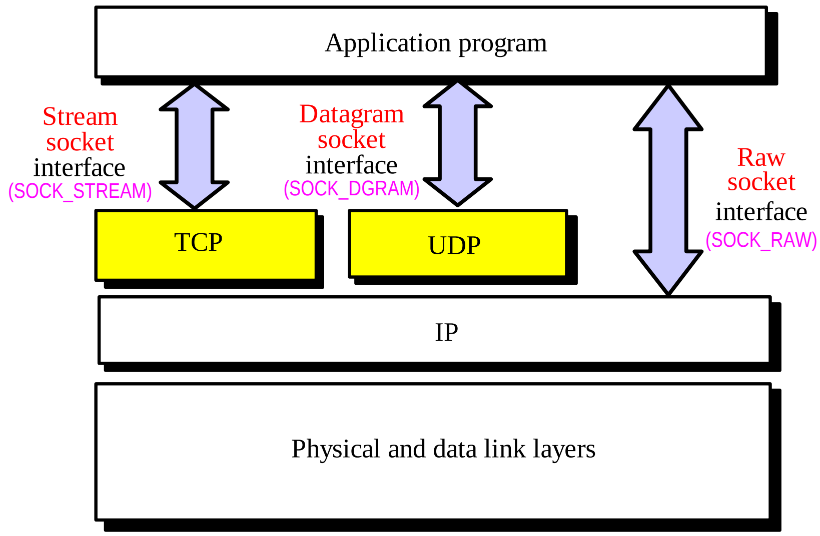
Socket API

- introduced in BSD4.1 UNIX, 1981
- explicitly created, used, released by apps
- client/server paradigm
- two types of transport service via socket API:
  - unreliable datagram
  - reliable, byte stream-oriented

socket

a *host-local*, *application-created/owned*, *OS-controlled* interface (a "door") into which application process can both send and receive messages to/from another (remote or local) application process

Socket types



2: Application Layer 13

Socket Functions

Server:	create endpoint	socket()
	bind address	bind()
	specify queue	listen()
	wait for connection	accept()
Client:	create endpoint	socket()
	bind address	bind()
	connect to server	connect()
	transfer data	read() write() recv() send()
	datagrams	recvfrom() sendto()
	terminate	close() shutdown()

2: Application Layer 14

socket() System Call

int socket (int *family*, int *type*, int *protocol*);

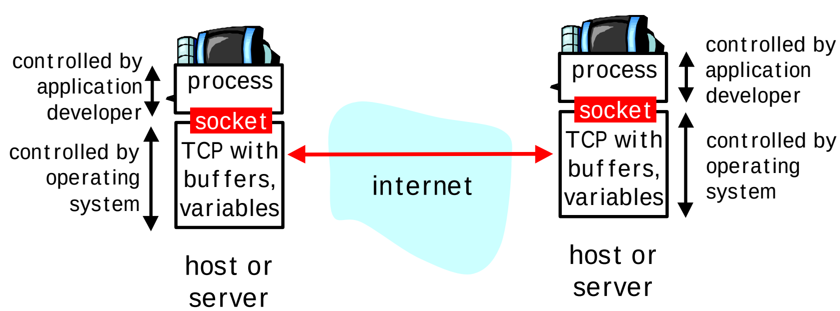
```
            AF_UNIX   SOCK_STREAM
            AF_INET   SOCK_DGRAM
                      SOCK_RAW
```

family	*type*	*protocol*	Actual protocol
AF_INET	SOCK_DGRAM	IPPROTO_UDP	UDP
AF_INET	SOCK_STREAM	IPPROTO_TCP	TCP
AF_INET	SOCK_RAW	IPPROTO_ICMP	ICMP
AF_INET	SOCK_RAW	IPPROTO_RAW	(raw)

Socket-programming using TCP

Socket: a door between application process and end-end-transport protocol (UCP or TCP)

TCP service: reliable transfer of bytes from one process to another



Socket programming with TCP

Client must contact server
- server process must first be running
- server must have created socket (door) that welcomes client's contact

Client contacts server by:
- creating client-local TCP socket
- specifying IP address, port number of server process

- When client creates socket: client TCP establishes connection to server TCP
- When contacted by client, server TCP creates new socket for server process to communicate with client
  - allows server to talk with multiple clients

application viewpoint

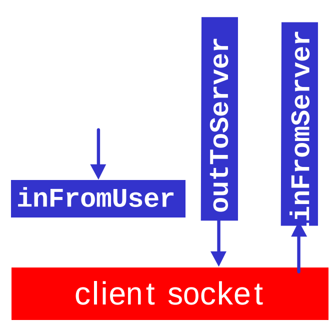
TCP provides reliable, in-order transfer of bytes ("pipe") between client and server

Socket programming with TCP

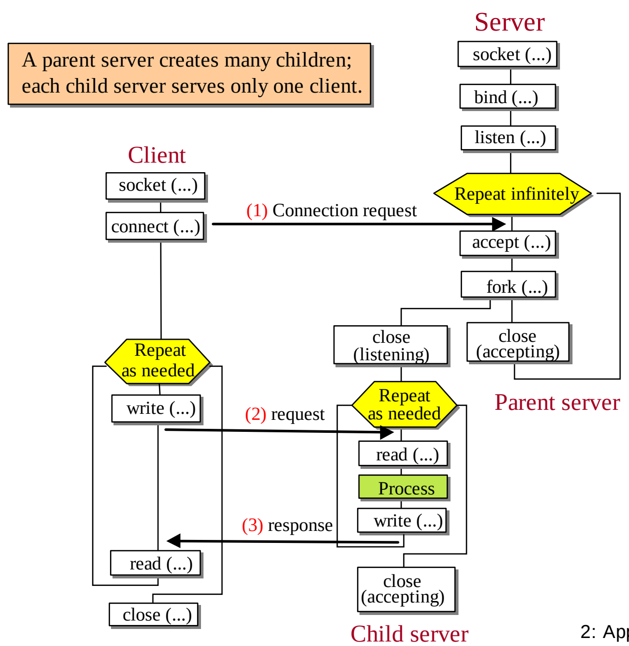
Example client-server app:
- client reads line from standard input (**inFromUser** stream) , sends to server via socket (**outToServer** stream)
- server reads line from socket
- server converts line to uppercase, sends back to client
- client reads, prints modified line from socket (**inFromServer** stream)

Input stream: sequence of bytes into process

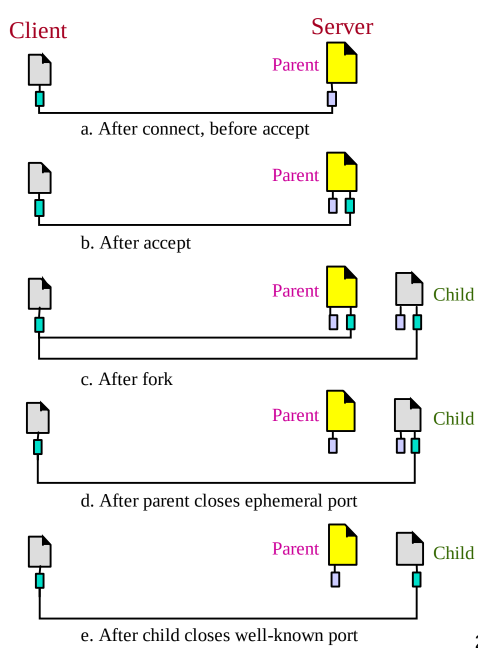
Output stream: sequence of bytes out of process



Socket interface for connection-oriented concurrent server



Relationship between the client and the server



TCP Concurrent Server Program

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdio.h>
#include <string.h>
#define MAXBUF 256
void main(void) {
   char buf[MAXBUF];
   int listenSocket;
   int acceptSocket;
   int clientAddrLen;
   struct sockaddr_in  serverAddr;
   struct sockaddr_in  cleintAddr;
   listenSocket = socket(AF_INET, SOCK_STREAM, 0);
   memset(&serverAddr, 0, sizeof(serverAddr));
   serverAddr.sin_family = AF_INET;
   serverAddr.sin_port = htons(a-well-know-port);
   serverAddr.sin_addr.s_addr = htonl(INADDR_ANY);
   bind(listenSocket, &serverAddr, sizeof(serverAddr));
   listen(listenSocket, 1);
   clientAddrLen = sizeof(clientAddr);
```

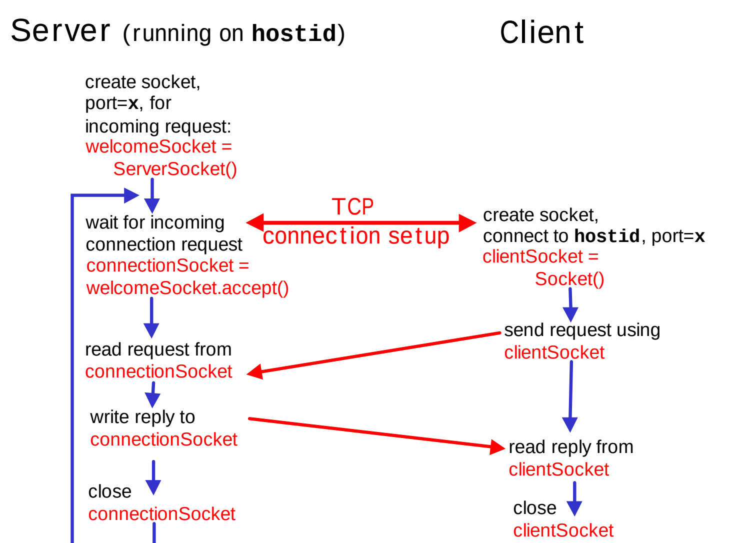

TCP Concurrent Server Program (cont'd)

```
      for (;;) {
         acceptSocket = accept(listenSocket, &clientAddr, &clientAddrLen);
         pid = fork();
         if (pid != 0) { /* parent */
            close(acceptSocket);
            continue;
         }  /* if */
         else {  /* child */
            close(lisetnSocket);
            memset(buf, 0, MAXBUF);
            while (read(acceptSocket, buf, MAXBUF) > 0) {
                 PROCESS (.........);
                 memset(buf, 0, MAXBUF);
                 write(acceptSocket, buf, MAXBUF);
                 memset(buf, 0, MAXBUF);
            }  /* while */
            close(acceptSocket);
         }  /* else */
      }  /* for */
   }  /* main */
```

TCP Concurrent Client Program

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdio.h>
#include <string.h>
#define MAXBUF 256
void main(void) {
   char buf[MAXBUF];
   int activeSocket;
   struct sockaddr_in  remoteAddr;
   struct sockaddr_in  localAddr;
   struct hostent  *hptr;
   activeSocket = socket(AF_INET, SOCK_STREAM, 0);
   memset(&remoteAddr, 0, sizeof(remoteAddr));
   remoteAddr.sin_family = AF_INET;
   remoteAddr.sin_port = htons(a-well-know-port);
   hptr = gethostbyname("a-domain-name");
   memcpy((char*)&remoteAddr.sin_addr.s_addr,
               hptr->h_addr_list[0], hptr->h_length);
   memset(&buf, 0, MAXBUF);
   while (gets(buf)) {
         write(activeSocket, buf, MAXBUF);
         memset(&buf, 0, MAXBUF);
         read(sockds, buf, MAXBUF);
         printf(%s\n, buf);
         memset(&buf, 0, MAXBUF);
      }  /* while */
      close(activeSocket);
   }  /* main */
```


Client/server socket interaction: TCP



Example: Java client (TCP)

```
import java.io.*;
import java.net.*;
class TCPClient {

    public static void main(String argv[]) throws Exception
    {
        String sentence;
        String modifiedSentence;

        BufferedReader inFromUser =
          new BufferedReader(new InputStreamReader(System.in));

        Socket clientSocket = new Socket("hostname", 6789);

        DataOutputStream outToServer =
          new DataOutputStream(clientSocket.getOutputStream());
```

Create input stream → BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));

Create client socket, connect to server → Socket clientSocket = new Socket("hostname", 6789);

Create output stream attached to socket → DataOutputStream outToServer = new DataOutputStream(clientSocket.getOutputStream());

Example: Java client (TCP), cont.

```
        BufferedReader inFromServer =
          new BufferedReader(new
          InputStreamReader(clientSocket.getInputStream()));

        sentence = inFromUser.readLine();

        outToServer.writeBytes(sentence + '\n');

        modifiedSentence = inFromServer.readLine();

        System.out.println("FROM SERVER: " + modifiedSentence);

        clientSocket.close();

    }
}
```

Create input stream attached to socket → BufferedReader inFromServer = ...

Send line to server → outToServer.writeBytes(sentence + '\n');

Read line from server → modifiedSentence = inFromServer.readLine();

Example: Java server (TCP)

```
import java.io.*;
import java.net.*;

class TCPServer {

 public static void main(String argv[]) throws Exception
  {
    String clientSentence;
    String capitalizedSentence;

    ServerSocket welcomeSocket = new ServerSocket(6789);

    while(true) {

       Socket connectionSocket = welcomeSocket.accept();

       BufferedReader inFromClient =
          new BufferedReader(new
          InputStreamReader(connectionSocket.getInputStream()));
```

Create welcoming socket at port 6789 → `ServerSocket welcomeSocket = new ServerSocket(6789);`

Wait, on welcoming socket for contact by client → `Socket connectionSocket = welcomeSocket.accept();`

Create input stream, attached to socket → `BufferedReader inFromClient = ...`

Example: Java server (TCP), cont

```
      DataOutputStream  outToClient =
        new DataOutputStream(connectionSocket.getOutputStream());

      clientSentence = inFromClient.readLine();

      capitalizedSentence = clientSentence.toUpperCase() + '\n';

      outToClient.writeBytes(capitalizedSentence);
     }
   }
 }
```

Create output stream, attached to socket → `DataOutputStream outToClient = ...`

Read in line from socket → `clientSentence = inFromClient.readLine();`

Write out line to socket → `outToClient.writeBytes(capitalizedSentence);`

End of while loop, loop back and wait for another client connection

Socket programming with UDP

UDP: no "connection" between client and server

- no handshaking
- sender explicitly attaches IP address and port of destination
- server must extract IP address, port of sender from received datagram

UDP: transmitted data may be received out of order, or lost

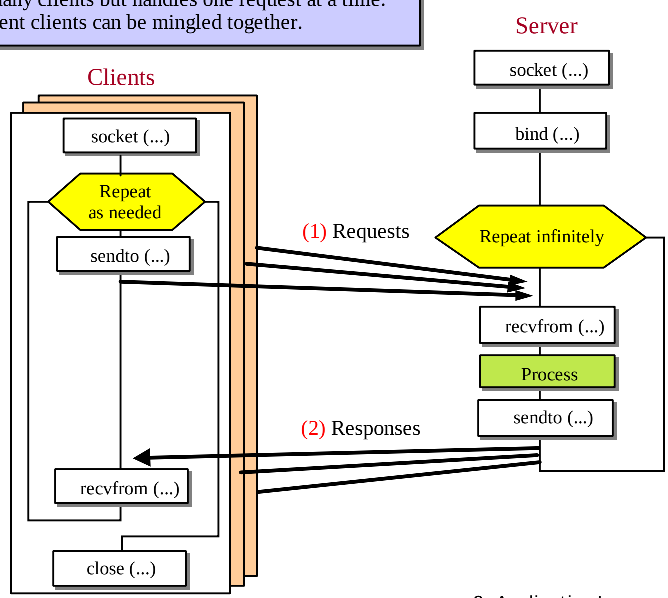
application viewpoint

UDP provides unreliable transfer of groups of bytes ("datagrams") between client and server

2: Application Layer 29

Socket interface for connectionless iterative server

Each server serves many clients but handles one request at a time. Requests from different clients can be mingled together.



2: Application Layer 30

UDP Iterative Server Program

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdio.h>
#include <string.h>
#define MAXBUF 256
void main(void)  {
   char buf[MAXBUF];
   int passiveSocket;
   int clientAddrLen;
   struct sockaddr_in  serverAddr;
   struct sockaddr_in  cleintAddr;
   passiveSocket = socket(AF_INET, SOCK_DGRAM, 0);
   memset(&serverAddr, 0, sizeof(serverAddr));
   serverAddr.sin_family = AF_INET;
   serverAddr.sin_port = htons(a-well-know-port);
   serverAddr.sin_addr.s_addr = htonl(INADDR_ANY);
   bind(passiveSocket, &serverAddr, sizeof(serverAddr));
   clientAddrLen = sizeof(clientAddr);
   for (;;) {
      while (recvfrom(passiveSocket, buf, MAXBUF,
               0, &clientAddr, &clientAddrLen) > 0)  {
         PROCESS (.........);
         memset(buf, 0, MAXBUF);
         sendto(passiveSocket, buf, MAXBUF, 0,
                  &clientAddr, clientAddrLen);
         memset(buf, 0, MAXBUF);
      }  /* while */
   }  /* for */
}  /* main */
```

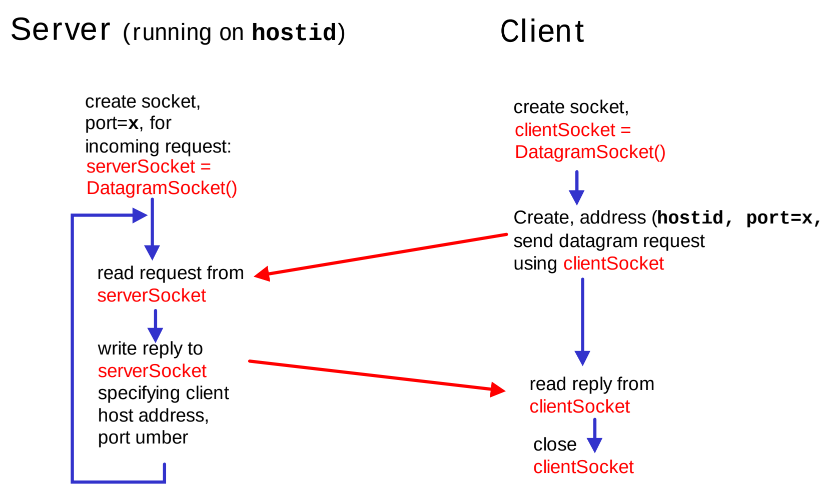
2: Application Layer 31

UDP Iterative Client Program

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdio.h>
#include <string.h>
#define MAXBUF 256
void main(void)  {
   char buf[MAXBUF];
   int activeSocket;
   struct sockaddr_in  remoteAddr;
   struct sockaddr_in  localAddr;
   struct hostent  *hptr;
   activeSocket = socket(AF_INET, SOCK_DGRAM, 0);
   memset(&remoteAddr, 0, sizeof(remoteAddr));
   remoteAddr.sin_family = AF_INET;
   remoteAddr.sin_port = htons(a-well-know-port);
   hptr = gethostbyname("a-domain-name");
   memcpy((char*)&remoteAddr.sin_addr.s_addr,
              hptr->h_addr_list[0], hptr->h_length);
   connect(activeSocket, &remoteAddr, sizeof(remoteAddr));
   memset(&buf, 0, MAXBUF);
   remoteAddLen = sizeof(remoteAddr);
   while (gets(buf)) {
      sendto(activeSocket, buf, size(buf), 0,
               &remoteAddr, sizeof(remoteAddr));
      memset(&buf, 0, MAXBUF);
      recvfrom(activeSocket, buf, MAXBUF, 0,
               &remoteAddr, &remoteAddrLen);
      printf(%s\n, buf);
      memset(&buf, 0, sizeof(buf));
   }  /* while */
   close(activeSocket);
}  /* main */
```

2: Application Layer 32

Client/server socket interaction: UDP



Example: Java client (UDP)

Create input stream → BufferedReader inFromUser
Create client socket → DatagramSocket clientSocket
Translate hostname to IP address using DNS → InetAddress IPAddress

```
import java.io.*;
import java.net.*;

class UDPClient {
  public static void main(String args[]) throws Exception
  {

    BufferedReader inFromUser =
      new BufferedReader(new InputStreamReader(System.in));

    DatagramSocket clientSocket = new DatagramSocket();

    InetAddress IPAddress = InetAddress.getByName("hostname");

    byte[] sendData = new byte[1024];
    byte[] receiveData = new byte[1024];

    String sentence = inFromUser.readLine();

    sendData = sentence.getBytes();
```

Example: Java client (UDP), cont.

Create datagram with data-to-send, length, IP addr, port

```
DatagramPacket sendPacket =
  new DatagramPacket(sendData, sendData.length, IPAddress, 9876);
```

Send datagram to server

```
clientSocket.send(sendPacket);

DatagramPacket receivePacket =
  new DatagramPacket(receiveData, receiveData.length);
```

Read datagram from server

```
clientSocket.receive(receivePacket);

String modifiedSentence =
   new String(receivePacket.getData());

System.out.println("FROM SERVER:" + modifiedSentence);
clientSocket.close();
}
}
```


Example: Java server (UDP)

```
import java.io.*;
import java.net.*;

class UDPServer {
  public static void main(String args[]) throws Exception
    {
```

Create datagram socket at port 9876

```
      DatagramSocket serverSocket = new DatagramSocket(9876);

      byte[] receiveData = new byte[1024];
      byte[] sendData  = new byte[1024];

      while(true)
        {
```

Create space for received datagram

```
          DatagramPacket receivePacket =
             new DatagramPacket(receiveData, receiveData.length);
```

Receive datagram

```
          serverSocket.receive(receivePacket);
```

Example: Java server (UDP), cont

```
        String sentence = new String(receivePacket.getData());

        InetAddress IPAddress = receivePacket.getAddress();

        int port = receivePacket.getPort();

                String capitalizedSentence = sentence.toUpperCase();

        sendData = capitalizedSentence.getBytes();

        DatagramPacket sendPacket =
          new DatagramPacket(sendData, sendData.length, IPAddress,
                             port);

        serverSocket.send(sendPacket);
      }
    }
}
```

Get IP addr port #, of sender → InetAddress IPAddress = receivePacket.getAddress(); / int port = receivePacket.getPort();

Create datagram to send to client → DatagramPacket sendPacket = ...

Write out datagram to socket → serverSocket.send(sendPacket);

End of while loop, loop back and wait for another datagram

Chapter 2: Summary

Our study of network apps now complete!

- application service requirements:
  - reliability, bandwidth, delay
- client-server paradigm
- Internet transport service model
  - connection-oriented, reliable: TCP
  - unreliable, datagrams: UDP
- specific protocols:
  - http
  - ftp
  - smtp, pop3
  - dns
- socket programming
  - client/server implementation
  - using tcp, udp sockets

Chapter 2: Summary

Most importantly: learned about *protocols*

- typical request/reply message exchange:
  - client requests info or service
  - server responds with data, status code
- message formats:
  - headers: fields giving info about data
  - data: info being communicated

- control vs. data msgs
  - in-based, out-of-band
- centralized vs. decentralized
- stateless vs. stateful
- reliable vs. unreliable msg transfer
- "complexity at network edge"
- security: authentication